INTRODUCTION TO DESIGN PATTERN

Bayu Priyambadha

INTRODUCTION

- Designing object-oriented software is hard, and designing reusable object-oriented software is even harder
  - Identify objects, factor them into classes at the right granularity, define class interfaces and inheritance hierarchies, and establish key relationships
  - The design should be specific to the problem at hand but also general enough to address future problems and requirements
- New designers tend to fall back on non-OO techniques used before
- Experienced designers know something – what is it?
  - Expert designers know not to solve every problem from first principles
  - They reuse solutions
- These patterns make OO designs more flexible, elegant, and ultimately reusable

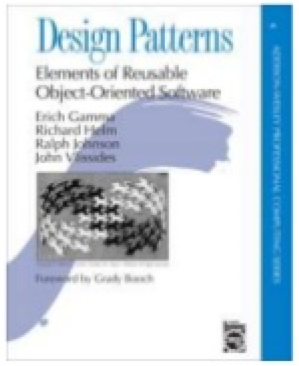
DESIGN PATTERN

"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice“ (Christopher Alexander)

Popularized by Gamma, Helm, Johnson and Vlissides (The gang of four, Go4, GOF)

Gang Of Four - GoF

- Ralph Johnson, Erich Gamma, Richard Helm, John Vlissides


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch

A DESIGN PATTERN

- Abstracts a recurring design structure
  - comprises class and/or object
  - dependencies
  - structures
  - interactions
  - conventions
- Names & specifies the design structure explicitly
- Distills design experience
- Is good, if it:
  - be as general as possible
  - contains a solution that has been proven to effectively solve the problem in the indicated context

DESIGN PATTERN TEMPLATE ELEMENTS

Name

- A meaningful name that reflects the knowledge embodied by the pattern

Problem

- Describes the problem that the pattern addresses

Context

- The general situation in which the pattern applies, including the application domain

Forces

- The issues or concerns to consider when solving the problem, including limitations and constraints

Solution

- The recommended way to solve the problem in the given context => should resolve all the forces

GOALS

- Codify good design
  - distill & generalize experience
  - aid to novices & experts alike
- Give design structures explicit names
  - common vocabulary
  - reduced complexity
  - greater expressiveness
- Capture & preserve design information
  - articulate design decisions succinctly
  - improve documentation
- Facilitate restructuring/refactoring
  - patterns are interrelated
  - additional flexibility

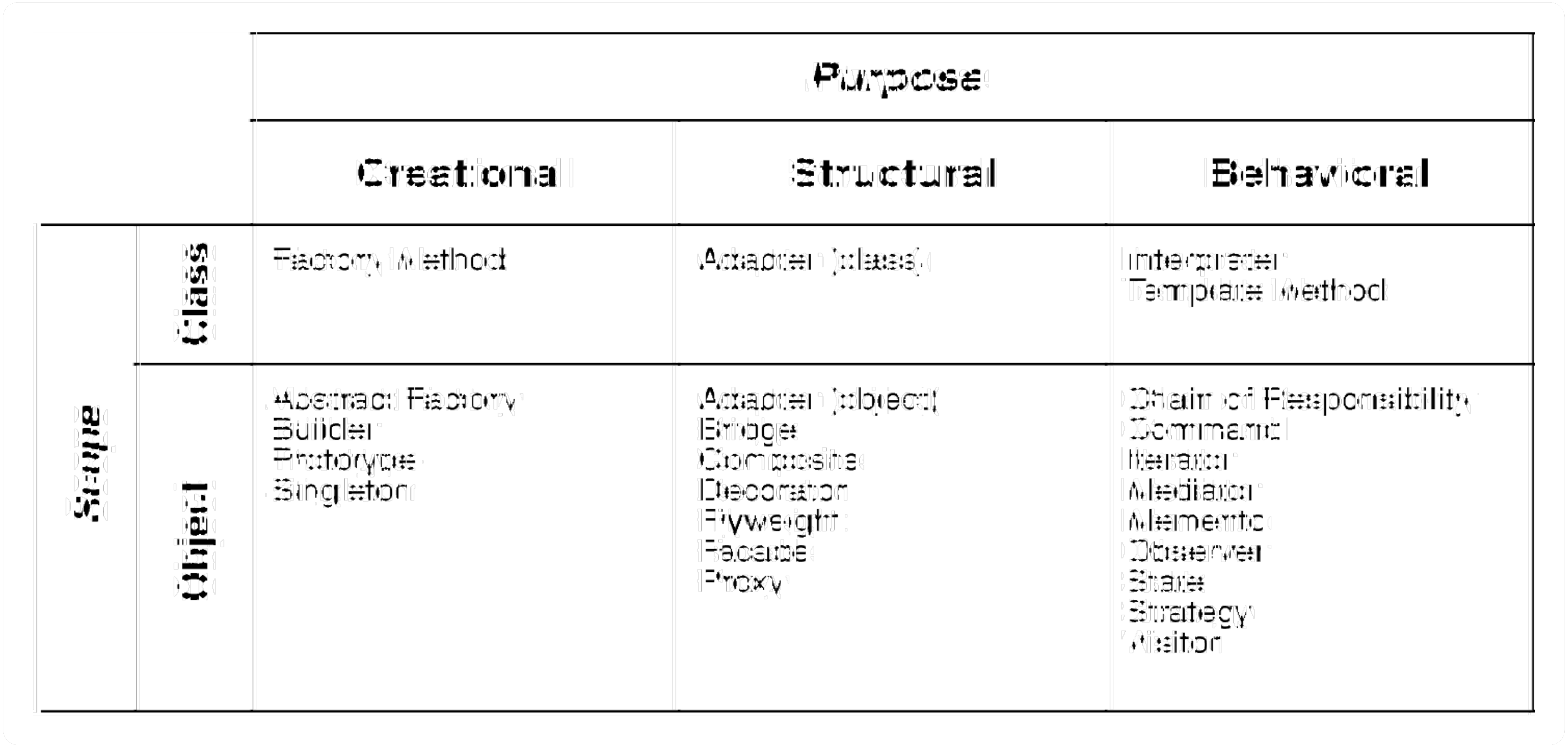
GOF’S CATEGORIES

- Based on scope (domain over which a pattern applies):
  - Class => concerns with class and the relationship to its sub-classes at the compile-time (static)
  - Object => concerns with objects and their relationships at the run-time (dynamic)

GOF’S CATEGORIES

- Based on purposes (reflects what a pattern does):
  - Creational => concerns with the construction of object instances
    - Class => defer its object creation to subclasses
    - Object => defer part of its object creation to another object
  - Structural => how objects are composed into larger groups
    - Class => structure via inheritance
    - Object => structure via composition
  - Behavioral => how responsibilities are distributed
    - Class => algorithms/control via inheritance
    - Object => algorithms/control via object groups/composition

TYPES



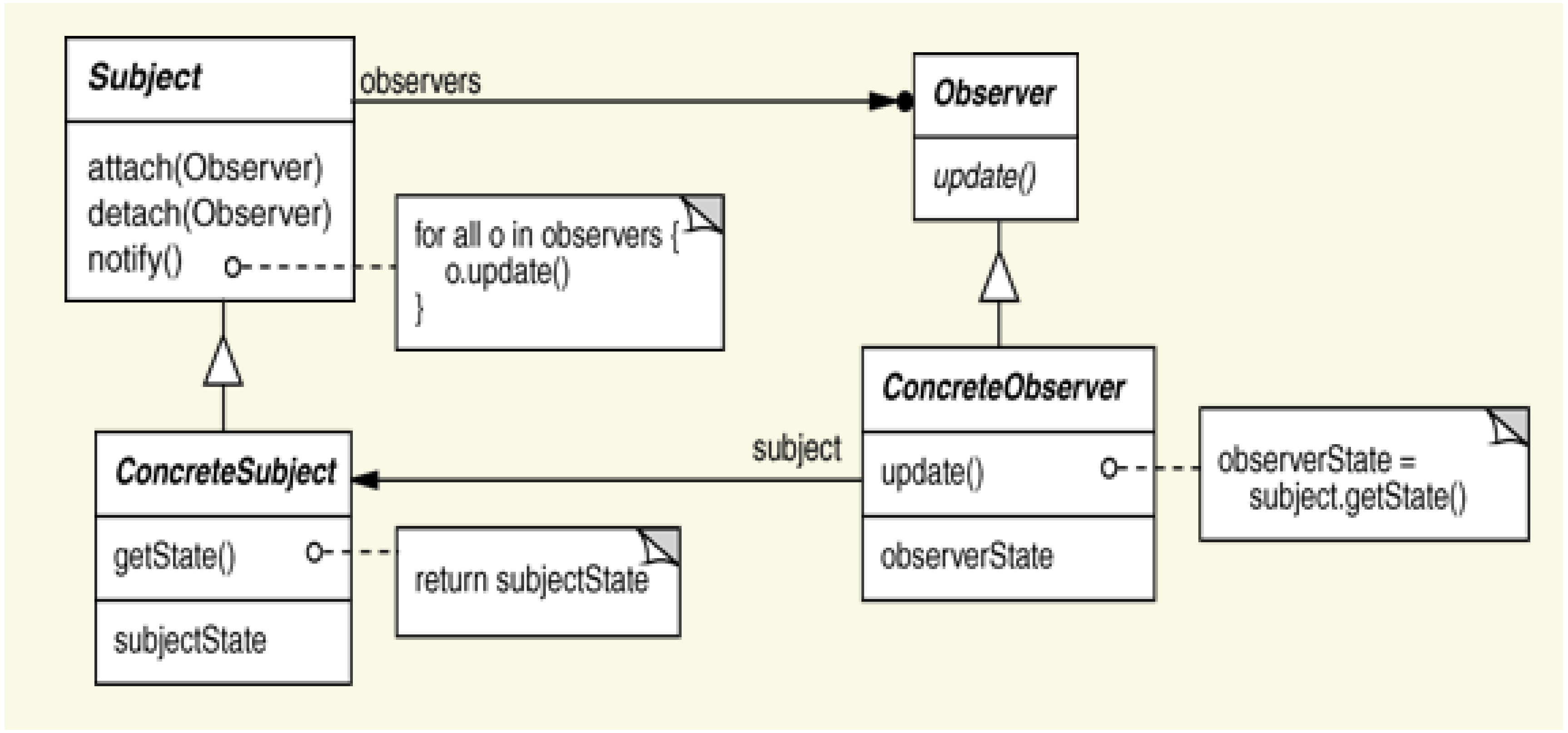
		Purpose		
		Creational	Structural	Behavioral
Scope	Class	Factory Method	Adapter (class)	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter (object) Bridge Composite Decorator Flyweight Facade Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

EXAMPLE: OBSERVER (BEHAVIORAL)

- Context:
  - When an association is created between two classes, the code for the classes becomes inseparable
  - If you want to reuse one class, then you also have to reuse the other
- Problem:
  - How do you reduce the interconnection between classes, especially between classes that belong to different modules or subsystems?
- Forces:
  - You want to maximize the flexibility of the system to the greatest extent possible

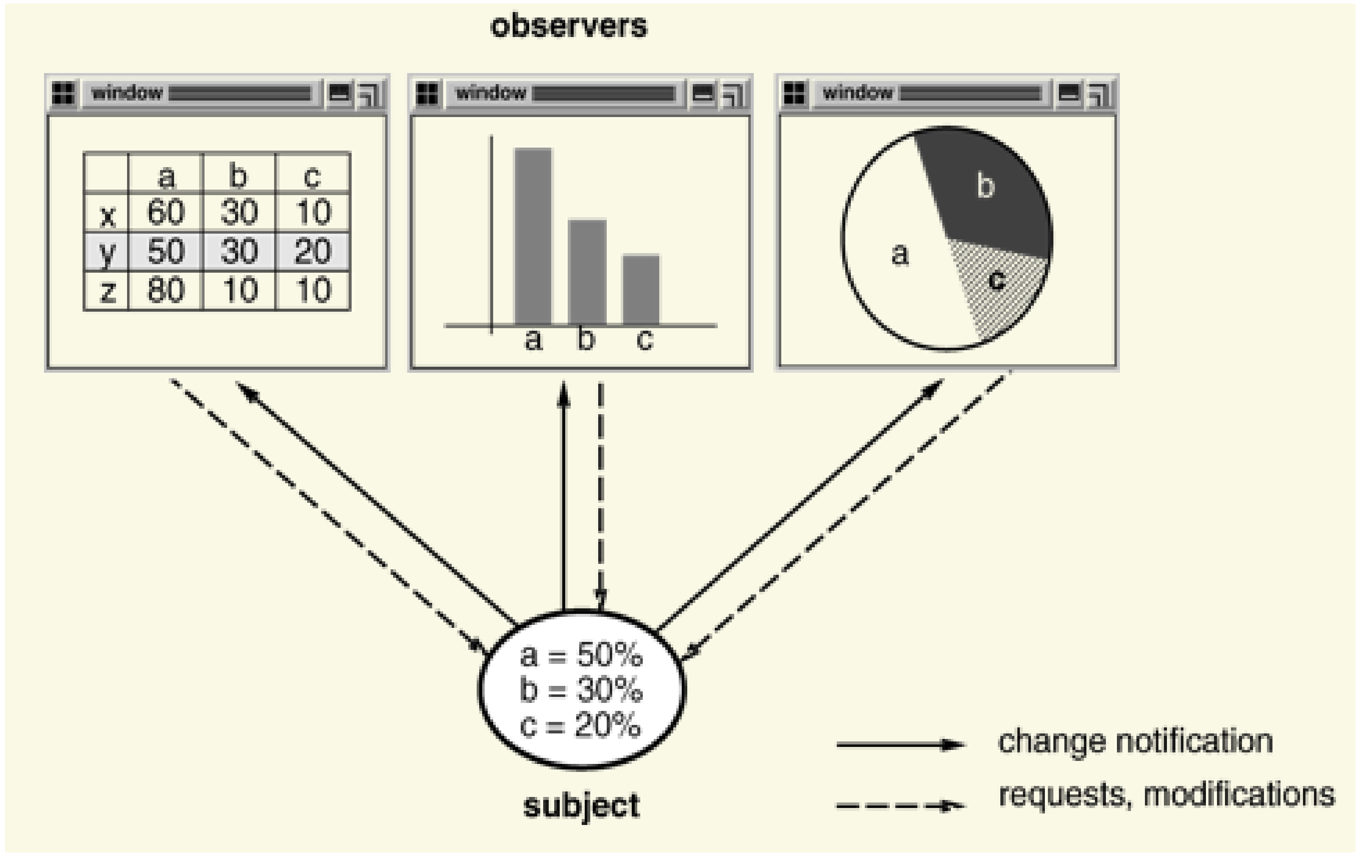
EXAMPLE: OBSERVER (BEHAVIORAL)

- Solution :



EXAMPLE: OBSERVER (BEHAVIORAL)

- Example :



BENEFITS AND DANGERS OF USING PATTERNS

- + Reuse of generic solutions
- + They provide a vocabulary for discussing the problem domain at a higher level of abstraction
- + Enhance understanding, restructuring, & team communication
- – May limit creativity
- – The use of patterns may lead to over-design
- Organizational impact
  - The use of patterns requires care and planning
  - Patterns must be used with intelligence

SUMMARY

- Patterns have been identified in many different application domains and are applicable at many different stages of the software development process
- Patterns are not a panacea:
  - Whenever you see an indication that a pattern should be applied, you might be tempted to blindly apply the pattern
  - This can lead to unwise design decisions
  - Always understand in depth the forces that need to be balanced, and when other patterns better balance the forces
  - Make sure you justify each design decision carefully

Thanks...